THE HONG KONG POLYTECHNIC UNIVERSITY
HONG KONG COMMUNITY COLLEGE

Subject Title	: Data Structures	**Subject Code**	: CCN2239
Session	: Semester One, 2017/18	**Time**	: 09:30 – 12:30
Date	: 21 December 2017	**Time Allowed**	: 3 Hours
Subject Examiner(s)	: Dr Pat CHAN Dr Joseph SO		

This question paper has a total of **TWENTY-ONE** pages (including this covering page).

Instructions to Candidates:

1. There are THREE sections in this paper.
   - Section A (30%) – Multiple-choice Questions. Answer ALL questions in this section on the multiple-choice answer sheet provided. Each question carries 1 mark. Choose the BEST option for each question.
   - Section B (50%) – Short Questions. Answer any FIVE out of the SIX questions in this section in the answer book provided. Each question carries 10 marks. If you answer more than five questions, only the first five attempted questions will be marked. Indicate in your answer book clearly which five questions you are attempting.
   - Section C (20%) – Long Questions. Answer any ONE out of the TWO questions in this section in the answer book provided. Each question carries 20 marks. If you answer more than one question, only the first attempted question will be marked. Indicate in your answer book clearly which one question you are attempting.
2. Candidates are NOT allowed to retain the multiple-choice answer sheet, the answer book and the examination question paper.
3. Show all your work clearly and neatly. Marks will be deducted for untidy work.
4. Reasonable steps should be shown.
5. All programming code must be written in Java programming language.

Authorised Materials:

	YES	NO
CALCULATOR	[ ]	[✓]
SPECIFICALLY PERMITTED ITEMS	[ ]	[✓]

DO NOT TURN OVER THE PAGE UNTIL YOU ARE TOLD TO DO SO

Section B (50%) – Short Questions

Answer any FIVE out of the SIX questions in this section in the answer book provided. Each question carries 10 marks. If you answer more than five questions, only the first five attempted questions will be marked. Indicate in your answer book clearly which five questions you are attempting.

Question B1

You are given the following java program.

```
public class Drawing {

      public int first(int m) {
            int u = 0, j= 0;
            while (j <= m) {

                  for (int i = 0; i <= j * 2; i++) {
                        System.out.print(i);
                        u += i;
                  }
                  System.out.println();
                  j++;
            }
            return u;
      }

      public int second(int m) {
            int v = 2, r = 0;
            for (int j = 1; j < m && v <= 200; j++) {

                  for (int k = 0; k < j; k+=2) {

                        r = k%3;
                        switch(k) {
                        case 2:
                              System.out.print("$");
                              v*=r;
                              break;
                        case 1:
                              System.out.print("^");
                        case 0:
                              System.out.print("0");
                              break;
                        }
                  }
                  System.out.println(v);

            }
            return v;
      }
```

Question B1 (continued)

```
    public int tryout(int m) {
        if (m <= 0)
            return 1;
        System.out.println(m);
        return tryout(m - 3) *2  + tryout(m - 1);

    }

    public static void main(String[] args) {
        Drawing p = new Drawing();
        System.out.println(p.first(4));
        System.out.println();
        System.out.println(p.second(6));
        System.out.println();
        System.out.println(p.tryout(6));
    }

}
```

Show the output when successfully executing `Drawing` in java. (10 marks)

Question B2

Sort the following sequence of keys by the following sorting algorithms with ascending order. Write down clearly the sequences in each pass.

```
99, 66, 77, 1, 11, 88, 100, 22, 33, 44, 55
```

(a) Selection sort. (4 marks)

(b) Bubble sort with early termination. (4 marks)

(c) Can *Selection Sort* execute early termination? Explain briefly. (2 marks)

Question B3

(a) What is the output of following program? (2 marks)

```
class AA3 {
        public int i;
        protected int j;
    }
    class BB3 extends AA3 {
        int j;
        void display() {
            super.j = 5;
            System.out.println(i + " " + j);
        }
    }
```

```
class testBB3 {
    public static void main(String args[])
    {
        BB3 obj = new BB3();
        obj.i=1;
        obj.j=3;
        obj.display();
    }
}
```

(b) You are given the following Java classes.

```
public class Year2017{
    public String toString() { return "2017"; }
}

public class Test2017 extends Year2017{
    public void print() {

        <missing statement>

    }
    public String toString() { return "2016"; }
}
```

Replace `<missing statement>` so that `Test2017` would be compiled with no errors and the below `test.print()` would display `2017`? (2 marks)

```
Test2017 test = new Test2017();
test.print();
```

(c) Given a 4-element stack S with elements 1, 3, 5, 7 (from top to bottom), and a queue Q with elements 2, 4 (from front to rear). The elements in S are removed one by one, and inserted into Q. Then, elements in Q are removed one by one, and re-inserted into S. What would be the sequence of the elements in S (from top to bottom)? (3 marks)

(d) The following sequence of operations is performed on a stack:

Push(1), Push(2), Pop, Push(3), Push(3), Pop, Push(6), Pop, Pop, Pop, Push(5), Pop.

What is the sequence of popped out values? (3 marks)

Question B4

(a) Define what a *min tree* is. (1 mark)

(b) Give the **TWO** defining characteristics of a max heap. (2 marks)

(c) State the smallest and largest total number of nodes found in a max heap with height equal to 6 (assume the height of a single-node heap is 1). (2 marks)

(d) Draw the heap tree obtained after the following values are inserted into the tree in order.

27, 43, 15, 34, 30, 10, 18, 36, 19

(5 marks)

Question B5

Suppose you have the following hash table, implemented using linear probing. The hash function we are using is the identity function, $h(x) = x \bmod 9$.

0	1	2	3	4	5	6	7	8
9	18		12	3	14	4	21	

(a) In which order (A-E shown below) could the elements have been added to the hash table? There are several correct answers, and you should give all of them. Assume that the hash table has never been resized, and no elements have been deleted yet. Give explanation for each wrong sequence. (5 marks)

A: 9, 14, 4, 18, 12, 3, 21
B: 12, 3, 14, 18, 4, 9, 21
C: 12, 14, 3, 9, 4, 18, 21
D: 9, 12, 14, 3, 4, 21, 18
E: 12, 9, 18, 3, 14, 21, 4

(b) What is collision in a hash table? (1 mark)

(c) Apart from linear probing, state **TWO** other methods to handle collision. (4 marks)

Question B6

(a) Draw the expression tree of the following infix expression. (4 marks)

```
( (A + B) + C * (D + E) + F ) * (G + H)
```

(b) Convert the expression in (a) into prefix expression. (3 marks)

(c) Convert the expression in (a) into postfix expression. (3 marks)

- End of Section B -

Section C (20%) – Long Questions

Answer any <u>ONE</u> out of the TWO questions in this section in the answer book provided. Each question carries 20 marks. If you answer more than one question, only the first attempted question will be marked. Indicate in your answer book clearly which one question you are attempting.

<u>Question C1</u>

(a) Figure 5 shows the linked list structure in which the object reference `head` points at the first node and object reference `pp` points at `head.next`.

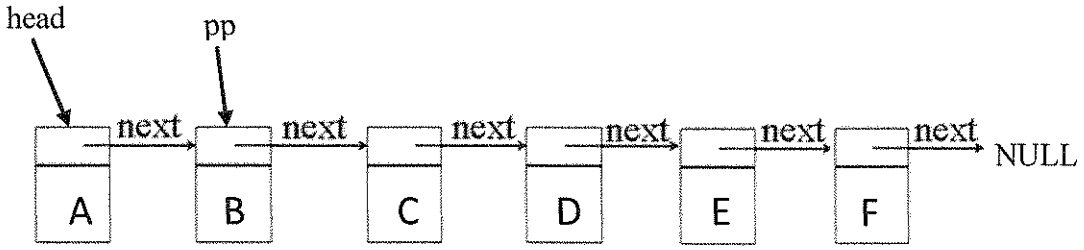

Figure 5

(i) After executing the following Java statements,

```
head.next = head.next.next;
pp.next = head.next.next;
head.next.next.next = pp.next.next;
head.next.next = null;
```

the linked list in the above will be changed. Draw all original nodes with the new links.

(4 marks)

(ii) Which node will be collected by garbage collector? (1 mark)

(iii) Starting from Figure 5, there is one more object reference `nm` defined. Complete the following Java statements such that after executing these statements, the linked list in Figure 5 will be changed into the two lists shown in Figure 6 and the object reference `head` and `pp` will point to the first node of each list. (Note: Modifying any of the following given parts of the Java statements or adding Java statements are **<u>NOT</u>** allowed.)

(5 marks)

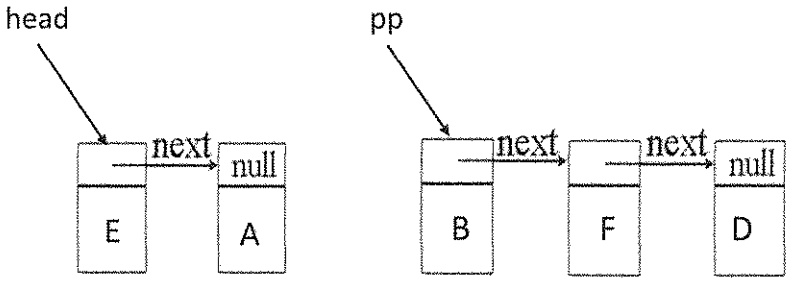

Figure 6

```
nm = head;
pp.next.next.next.next.next =
head =
head.next =
pp.next =
head.next.next =
pp.next.next.next =
```

(b) Given the following three Java classes, show the output when successfully executing `LListTest01` in java. (10 marks)

```
public class LListTest01 {

      public static void main(String[] args) {
            ChainTT lq1 = new ChainTT();

            lq1.add(0, new String("AA"));
            lq1.add(1, new Integer(6));
            lq1.add(0, new String("BB"));
            lq1.add(2, new Integer(4));

            System.out.println("List size is " + lq1.size());
            System.out.println("The list is " + lq1);

            System.out.println(lq1.remove(1) + " removed");
            lq1.add(2, new String("CC"));
            System.out.println(lq1.remove(1) + " removed");
            lq1.add(3, new String("DD"));
            lq1.add(3, new String("EE"));

            System.out.println("List size is " + lq1.size());
            System.out.println("The list is " + lq1);

      }
}

class ChainNode
{

    Object element;
    public ChainNode next;

    ChainNode() {}

    ChainNode(Object element, ChainNode next)
       {this.element = element;
        this.next = next;}
}
```

Question C1 (continued)

```
import java.util.*;

public class ChainTT
{

   protected ChainNode firstNode;
   protected int size;

   public int size()
      {return size;}

   public Object get(int index)
   {
      checkIndex(index);

      // move to desired node
      ChainNode currentNode = firstNode;
      for (int i = 0; i < index; i++)
         currentNode = currentNode.next;

      return currentNode.element;
   }

   public int indexOf(Object theElement)
   {
      // search the chain for theElement
      ChainNode currentNode = firstNode;
      int index = 0;  // index of currentNode
      while (currentNode != null &&
            !currentNode.element.equals(theElement))
      {
         // move to next node
         currentNode = currentNode.next;
         index++;
      }

      // make sure we found matching element
      if (currentNode == null)
         return -1;
      else
         return index;
   }

   public Object remove(int index)
   {
      checkIndex(index);

      Object removedElement;
      if (index == 0) // remove first node
      {
         removedElement = firstNode.element;
         firstNode = firstNode.next;
      }
      else
      {  // use q to get to predecessor of desired node
```

Question C1 (continued)

```
            ChainNode q = firstNode;
            for (int i = 0; i < index - 1; i++)
               q = q.next;

            removedElement = q.next.element;
            q.next = q.next.next; // remove desired node
         }
         size--;
         return removedElement;
      }

       public void add(int index, Object theElement)
      {
         if (index < 0 || index > size)
            throw new IndexOutOfBoundsException
                  ("index = " + index + "  size = " + size);

         if (index == 0)
            firstNode = new ChainNode(theElement, firstNode);
         else
         {
            ChainNode p = firstNode;
            for (int i = 0; i < index - 1; i++)
               p = p.next;

            p.next = new ChainNode(theElement, p.next);
         }
         size++;
      }

       void checkIndex(int index)
       {
          if (index < 0 || index >= size)
             throw new IndexOutOfBoundsException
                   ("index = " + index + "  size = " + size);
       }

      public String toString()
      {
         String s = new String("[");

         ChainNode currentNode = firstNode;
         while(currentNode != null)
         {
            if (currentNode.element == null)
               s +="null, ";
            else
               s += currentNode.element.toString() + " ";
            currentNode = currentNode.next;
         }
         s+= "]";

         return s;
      }
   }
```

Question C2

Suppose you have a BSTNode class that stores integers:

```
public class BSTNode {
      int element;
      BSTNode leftChild;
      BSTNode rightChild;

      public BSTNode(int x){
            element = x;
      }
}
```

The Java program IntegerBST.java shown below constructs a binary search tree with integers stored in nodes of type BSTNode. You are asked to complete this program by answering the question in parts (a) to (d).

```
import java.util.Random;

public class IntegerBST {

      private BSTNode root;

      public IntegerBST(){}

      public boolean put(int val){

            // <3>: Codes to be completed in part (c)
      }

      public boolean remove(int val){
            BSTNode p = root, pp = null;  // pp is parent of p

            // Instruction for part (d):
            // set p to point to the node with val
            // pp to point to its parent
            // if val is not found, p points to null

            // <4>: Codes to be completed in part (d)

            if (p == null) return false;

            // handle case when p has two children
            if (p.leftChild != null && p.rightChild != null){

                  BSTNode s, ps;    // ps is parent of s

            s = p.leftChild;
            ps = p;
            while (s.rightChild != null){
                  ps = s;
                  s = s.rightChild;
            }
            p = s;
```

Question C2 (continued)

```
                pp = ps;

                p.element = s.element;
        }

        BSTNode c;
        if (p.leftChild == null)
                c = p.rightChild;
        else
                c = p.leftChild;

        // remove node p
        if (p == root) root = c;
        else {
                if (p == pp.leftChild)
                        pp.leftChild = c;
                else
                        pp.rightChild = c;
        }
        return true;
}

public boolean search(int val){
```

// <2>: Codes to be completed in part (b)

```
}

private void print(int val){
        System.out.print(val + " ");
}

private void inOrder(BSTNode n){
```

// <1>: Codes to be completed in part (a)

```
}

public void printInOrder(){
        inOrder(this.root);
}

public static void main(String[] args) {
        IntegerBST tree = new IntegerBST();
        Random ran = new Random();
        for (int i = 0; i < 10; i++){
                int num = 1 + ran.nextInt(20);
                while (tree.put(num) == false){
                        num = 1 + ran.nextInt(20);
                }
        }
        System.out.println("Original list: ");
        tree.printInOrder();
        System.out.println();
        // remove nodes with multiple of 3 as values
        for (int i = 3; i < 20; i += 3){
```

Question C2 (continued)

```
                tree.remove(i);
            }
            System.out.println("New list: ");
            tree.printInOrder();
            System.out.println();
            // search nodes with multiple of 4 as values
            for (int i = 4; i <= 20; i += 4) {
                System.out.print(i + " is ");
                if (!tree.search(i)) System.out.print("not ");
                System.out.println("found in the tree.");
            }
        }
}
```

Sample output

```
Original list:
1 2 3 5 6 9 11 14 16 19
New list:
1 2 5 11 14 16 19
4 is not found in the tree.
8 is not found in the tree.
12 is not found in the tree.
16 is found in the tree.
20 is not found in the tree.
```

(a) By using **recursive** approach and calling the given `print` method, fill in your codes in blank <1> to complete the method `inOrder`, which prints the numbers in the binary tree in ascending order when it is called by `printInOrder`. (4 marks)

(b) Fill in your codes in blank <2> to complete the method `search` that returns true if the input argument *val* is found in the binary search tree, and returns false if otherwise. (3 marks)

(c) Fill in your codes in blank <3> to complete the method `put`. The method puts a new node storing the input argument *val* into the binary search tree and returns true if *val* is not found in the tree. If *val* is found in the tree, no new node is added and the method returns false. (10 marks)

(d) With reference to the instruction in the program's comment part, fill in your codes in blank <4> to complete the method `remove`. (3 marks)

- End of Section C -

- END OF PAPER -